ELSEVIER

Available online at www.sciencedirect.com
SCIENCE DIRECT®

Discrete Applied Mathematics 131 (2003) 169–177

DISCRETE APPLIED MATHEMATICS

www.elsevier.com/locate/dam

The task allocation problem with constant communication

W. Fernandez de la Vega, M. Lamari*

Universite de Paris Sud, Laboratorie de Recherche en Informatique, CNRS, UA 410, Centre d'Orsay, 91405 Orsay, France

Received 20 July 2000; received in revised form 17 May 2001; accepted 5 May 2002

Abstract

In the module allocation problem we are given n tasks $t_1,\ldots,t_n$, to be executed by m processors $P_1,\ldots,P_m$, subject to both execution and communication costs. The cost of any assignment of the tasks to the processors is defined as the sum of the corresponding execution costs, and the communication costs for any pair of tasks assigned to distinct processors. We consider the case where all the tasks communicate with communication costs all equal to a constant c_0.

When the number of processors is bounded, we give two exact, polynomial-time algorithms, an elementary one for the case where the execution costs take only two distinct values and one for the general case.

When the number of processors is not bounded, we obtain a polynomial-time approximation scheme.

We obtain a similar algorithm when the communication graph is the edge union of a bounded number of cliques and complete bipartite graphs.

Keywords: Scheduling; Polynomial time approximation scheme

1. Introduction

In the task allocation problem we are given n tasks $t_1,\ldots,t_n$, to be executed by m processors $P_1,\ldots,P_m$, subject to both execution and communication costs. We consider only *preemptive* schedules in which each task is executed on a single processor. The cost of any assignment of the tasks to the processors is defined as the sum of the

* Corresponding author.

0166-218X/03/$ - see front matter © 2003 Published by Elsevier B.V.
PII: S0166-218X(02)00423-7

corresponding execution costs, and the communication costs for any pair of tasks assigned to distinct processors.

This problem is also known as the module allocation problem (see, among others, Billionnet et al. [1], and Stone [10]). It has been proved to be $\mathcal{NP}$-hard by Magirou and Milis [7]. Their proof uses an equivalence, due to Stone [10], between the task assignment problem and *Multiway Cut*, which was proved to be hard by Dalhaus et al. [3].

The reduction between the task assignment problem and Multiway Cut suggests, of course, to use (approximate) Multiway Cut algorithms for the task assignment problem. However, this requires at first sight not only the usual (Karp) reduction but the stronger approximation-preserving L-reduction of Papadimitriou and Yannakakis [9]. We do not know if this kind of reduction is valid in the present case.

Several heuristics have been proposed [8,6]. Stone [10] gave a polynomial-time algorithm for the case of two processors. Bokhari [2] showed that the case where the graph of the communicating tasks, which we call the communication graph, is a tree, can be solved exactly using dynamic programming. Also, some random examples were shown to be tractable or approximable by Lamari and Fernandez de la Vega [5]. Here, we identify new cases of solvability, or approximability within $1+\varepsilon$, which concern communication graphs with a simple structure.

We consider mainly the case where the communication graph is the complete graph (i.e., every pair of tasks communicate) and moreover, the communication costs are all equal to a fixed constant. We also consider the case where the communication graph is the union of a bounded number of not necessarily disjoint cliques. (In fact, we consider a slightly more general case.) Here is a practical justification for considering a union of cliques: One may think of each task as possibly having some attributes from a finite set. If we assume that two tasks communicate iff they share at least one common attribute, then we get a communication graph in the above class. Concerning the execution costs, we consider either the general case or the bi-valued case in which, for each pair (t_i, P_j), the execution cost e_{ij} of task t_i on processor P_j satisfies $e_{ij} \in \{a, b\}$ where a and b are arbitrary fixed numbers with $0 \leqslant a < b$.

Our main results are the following. Recall that a polynomial-time approximation scheme for a given minimization problem is a family of algorithms $(A_\varepsilon)_{\varepsilon>0}$ such that, for each fixed ε, A_ε runs in polynomial time (on the size of the input) and provides a solution within a factor at most $1+\varepsilon$ from the optimum.

- When the number of processors is bounded, we give two exact, polynomial-time algorithms, an elementary one for the case where the execution costs take only two distinct values and one for the general case, based on the theory of flows.
- When the number of processors is not bounded, we obtain a polynomial-time approximation scheme.
- Then, we extend these results to the case where the communication graph is a union of cliques and then, rather directly, to the case where the communication graph is a union of pairwise disjoint cliques and complete bipartite graphs.

  All except the first (complete communication graph and bi-valued costs) of these results are obtained via the same procedure:

- Fix (exactly or approximately) a value of the communication cost, say C, (actually C will be fixed through the loads of the processors), and compute the minimum value of the execution cost, say $E = E(C)$, for the assignments corresponding to these loads.
- Search exhaustively for the loads for which the overall cost $C + E(C)$ is (approximately) minimum.

The justification of this procedure lies, of course, in the a posteriori fact that it works (within some limits). The a priori justification is that, communication and execution costs being of so dissimilar kinds (one is linear, the other quadratic), it seems hard to control them simultaneously.

Let us denote by T the set of tasks and by $\mathscr{P}$ the set of processors. In the sequel, for any assignment $A : T \to \mathscr{P}$ of the tasks to the processors, we will denote by $Exec(A)$ and $Com(A)$ the execution and communication costs corresponding to A. The overall cost of A will be denoted by $Cost(A)$:

$$Cost(A) = Exec(A) + Com(A).$$

2. Complete communication

Note first that the communication cost for an assignment A satisfies, under the assumption that every communication cost is equal to c_0, say,

$$Com(A) = \frac{c_0}{2} \sum_{i=1}^{m} n_i(n - n_i), \tag{1}$$

where $n_i = |A^{-1}(P_i)|$ is the load of processor P_i in the assignment A.

2.1. The case of bi-valued execution costs

We assume first that the execution costs are bi-valued: $e_{ij} \in \{a, b\}, 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m$ for fixed a and b with $0 \leqslant a < b$. Actually, there is no change if we assume $a = 0$.

In the case of bi-valued execution costs, we give an elementary algorithm, while our algorithm for the general case uses the theory of flows.

The algorithm will result from the following propositions.

Proposition 1. *In an optimal allocation A_{opt}, only the processor with maximum load (if any) executes tasks at cost b.*

Proof. Let $P_{j(1)}$ be the processor with maximum load and suppose for the proof that there is another processors $P_{j(k)}$ which also executes tasks with cost b in A_{opt}. Let n_1 (resp. n_k) denote the load of $P_{j(1)}$ (resp. $P_{j(k)}$). Thus we have $n_1 > n_k$. Consider now the assignment A' obtained from A_{opt} by moving to $P_{j(1)}$, one task executed on $P_{j(k)}$ at

cost b. This transfer certainly does not increase the total execution cost. The variation of the communication cost is, according to (1),

$$\Delta(Com) = c_0(n_k - 1 - n_1),$$

which is strictly negative, implying a contradiction. □

Let us denote by B the set of tasks whose execution costs are equal to b on all the processors. Thus, as far as the execution cost is concerned, we would like to execute only the tasks in B at cost b. However, in an optimal assignment it may be the case that some tasks not included in B are executed at cost b. We let B' denote the set of these tasks. We have then:

Proposition 2. *Let A_{opt} be an optimal assignment and let $P_{i(1)}, P_{i(2)}, \ldots, P_{i(m)}$, be the sequence of processors arranged according to their non-increasing loads $q_1, q_2, \ldots, q_m$ in tasks at cost a in A_{opt}. Then, the sequence $q_1, q_2, \ldots, q_m$ is lexicografically maximum among all possible sequences corresponding to this sequence of processors.*

Proof. Let t denote a task with execution cost a on both processors $P_{i(k)}$ and $P_{i(h)}$ with $h < k$. Check that if t is assigned to $P_{i(k)}$, then moving it to $P_{i(h)}$ diminishes the communication cost (and keeps the execution cost constant). □

The algorithm

The algorithm follows easily from Propositions 1 and 2. Note that there is only one lexicografically maximal sequence $q_1, q_2, \ldots, q_m$ of loads at cost a corresponding to a fixed sequence of processors. Therefore, if we know the optimal sequence of processors and the size s of the set B', then we can determine the optimum solution. We compute a lexicografically maximum assignment of the tasks at cost a on $P_{i(1)}, P_{i(2)}, \ldots, P_{i(m)}$, and then we transfer to $P_{i(1)}$ precisely s tasks from the rightmost processors: we transfer first tasks from $P_{i(m)}$ and then, if s exceeds the load of $P_{i(m)}$, we transfer tasks from $P_{i(m-1)}$, and so on. Note that, by proceeding this way, the remaining sequence of loads at cost a is also lexicografically maximum. Of course, we do not know the optimal sequence of processors, and thus we must apparently try separately each of the $m!$ distinct possibilities.

For any given sequence of processors we need at most $\lceil \log_2 m \rceil$ operations to find the corresponding assignment for any given task. This gives a total number of operations $O(nm! \log_2 m)$ and it is easy to see that the number of operations needed to cope with B' has a smaller order of magnitude.

Remark. One may wonder if it is indeed necessary to consider all sequences of processors in order to find the optimum assignment. Claire Kenyon found an example in which the sequence of processors giving the lexicografically maximum sequence of loads at cost a, which is of course a natural candidate to optimality, is in fact not optimal. (Here the maximum is taken with respect to every possible sequence of processors and this amounts to select first the processor which executes a maximum number of

tasks at cost a, then the one which executes the maximum number of remaining tasks at cost a, and so on).

3. Complete communication and arbitrary execution costs

In this section, we consider the case of communication costs all equal to a constant c_0, and arbitrary execution costs.

Let us first recall the *Minimum-Cost Flow Problem*. (See for instance [4, Problem (8.3.13), p. 238].)

Minimum-Cost Flow Problem. *Given a digraph $D=(V,A)$, two distinguished nodes $r,s\in V$, a capacity function $d:A\to Q_+$, a cost function $c:A\to Q_+$, and a rational number t, find an (r,s)-flow x, subject to the capacity function d, of value t, and with minimum cost $\sum_{a\in A} c_a x_a$.*

Here c_a and x_a stand for the unitary cost of the arc a and the flow through this arc, respectively. It is well known that a solvable minimum-Cost Flow Problem has an integer solution for integer t and d and that such a solution can be obtained in polynomial time. We will make use several times of this well-known result in the sequel.

3.1. The case of a bounded number of processors

We assume first that the number m of processors is bounded, say $m\leqslant m_0$, m_0 fixed. Let us fix the load n_i of each processor P_i (with $\sum_{i=1}^{m} n_i = n$). Then, an optimal assignment corresponding to these loads can be obtained by finding a minimum-cost integer flow with value n in a network which we proceed to describe.

Network 1: There are two kinds of nodes apart from the source and the sink: To each task T_i, there corresponds a node θ_i which is connected to the source by an arc with unit capacity. To each processor P_j, there corresponds a node p_j connected to the sink by an arc of capacity n_j. Moreover, each task node θ_i is connected to each processor node p_j by an arc with capacity 1 and cost e_{ij}.

Let now A be an assignment corresponding in the obvious way to an integer flow Φ in this network with value n and minimum cost. Clearly, the execution cost of A is minimum within all assignments giving the loads n_i and we have $Exec(A)=cost(\Phi)$. The overall cost of A is thus

$$Cost(A)=cost(\Phi)+c_0\sum_{1\leqslant i<j\leqslant m} n_i n_j.$$

The following procedure suggests itself immediately: try all the systems of loads $n_1,\ldots,n_m$ (with $\sum_{i=1}^{m} n_i=n$) and detect the one ('s) which results in an assignment A with minimum global cost. Observe that the number of choices for the n_i's is bounded

above by the number of ordered partitions of n into m parts. This number does not exceed n^{m-1} while an integer flow with minimum cost can be computed in time $O(n^{5/2})$. The time complexity of the procedure is thus bounded above by $n^{m_0+3/2}$.

3.2. The case where the number of processors is not bounded

When the number of processors is not bounded, we have not succeeded in finding an efficient exact algorithm. We have obtained, however, a polynomial-time approximation scheme.

Fix $\varepsilon > 0$. Let us define for any assignment A the index sets $I = \{i\colon\ n_i > \varepsilon n\}$ and $J = \{i\colon\ n_i \leqslant \varepsilon n\}$, where, for $1 \leqslant i \leqslant m$, $n_i = |A^{-1}(P_i)|$ denotes the load of P_i. Let $s = \sum_{i\in J} n_i$ and let us write

$$\bar{C}om(A) = \bar{C}om(\varepsilon, A) = \frac{c_0}{2}\sum_{i\in I} n_i(n - n_i) + \frac{c_0}{2}s(n-1).$$

$\bar{C}om(A)$ differs from $Com(A)$ exactly in that it counts the pairs of vertices $\{x, y\}$ where x and y are assigned to a same processor with load $\leqslant \varepsilon n$. We have thus

$$\bar{C}om(A) \leqslant Com(A) + c_0(\varepsilon n - 1)\sum_{i\in J} n_i. \tag{2}$$

We have also clearly,

$$Com(A) \geqslant \frac{c_0}{2}(n - \varepsilon n)\sum_{i\in J} n_i. \tag{3}$$

Now, (2) and (3) give

$$\frac{\bar{C}om(A)}{Com(A)} \leqslant 1 + \frac{2\varepsilon}{1-\varepsilon} \leqslant 1 + 3\varepsilon \tag{4}$$

for $\varepsilon \leqslant \frac{1}{3}$, while of course $\bar{C}om(A) \geqslant Com(A)$.

We call the processors whose loads do not exceed εn the tail processors. The other processors are called the head processors. Clearly, the number of head processors is at most $h = \lfloor \varepsilon^{-1} \rfloor$. We shall put, for any assignment A,

$$\bar{C}ost(A) = Exec(A) + \bar{C}om(A).$$

If we knew in advance the head processors and their loads, inequality (4) would clearly lead to an algorithm with approximation ratio $\leqslant 1 + 3\varepsilon$. It suffices thus to consider all the possibilities for the head processors $P_1, \ldots, P_m$, say, and for their loads $n_1, \ldots, n_m$, and select one which leads to a minimum overall cost. For each such system, the corresponding execution cost is the minimum-cost integer flow with value n in the following network.

Network 2: The task nodes are connected to the source and to the processor nodes exactly as in Network 1. Moreover, each head processor P_i is connected to the sink by an arc of capacity n_i and each tail processor is connected to an intermediate vertex

v by an arc of capacity εn. The vertex v is connected to the sink by an arc of capacity $n - \sum_{i=1}^{h} n_i$. This concludes the description of the network.

Consider now a system of loads n_i corresponding to an optimal assignment A_{opt}, and let A be some assignment corresponding to a minimum cost flow in the network with the loads n_i. Clearly, because of (4), A yields an approximation with ratio at most $1 + 3\varepsilon$ to the original problem. It suffices thus to try all the possible systems of head processors and loads of these processors.

The number of choices for the head processors is bounded above by $m^h/h!$. The number of choices for the loads of these processors is bounded above by the number $\binom{n-1}{h}$ of ordered partitions of n into $h+1$ parts. The overall time complexity is thus, up to a constant factor, bounded above by

$$n^{5/2} \frac{m^h}{h!} \frac{n^h}{h!} h! \leqslant n^{5/2} \frac{(mn)^h}{h!}.$$

Since $h \leqslant 1/\varepsilon$, we obtain the time complexity bound

$$\varepsilon^{1/2-1/\varepsilon} n^{5/2+1/\varepsilon} (me)^{1/\varepsilon}.$$

3.3. The case of several cliques

We assume now that the communication graph is the union of a bounded number of not necessarily disjoint cliques. We consider only the case of a bounded number of processors, say $m \leqslant m_0$. The case of an unbounded number of processors is similar to the case treated in the preceding section and is omitted.

Let $S_1, \ldots, S_k$ be (the vertex sets of) the cliques defining the communication graph. There is then a minimal family of pairwise disjoint sets of tasks, $\mathcal{T} = \{T_1, \ldots, T_h\}$, say, with the property that each S_i is the union of a subfamily of $\mathcal{T}$. Note that we have $h \leqslant 2^k - 1$. For instance, for $k=2$ and $S_{12} = S_1 \cap S_2 \neq \emptyset$, we can write $T_1 = S_1 \setminus S_{12}$, $T_2 = S_{12}$, and $T_3 = S_2 \setminus S_{12}$. Let $n_i = |T_i|, 1 \leqslant i \leqslant h$. Let us introduce the graph $G = (V(G), E(G))$ with vertex set $V(G) = \{1, 2, \ldots, h\}$ and where two vertices i and j are linked by an edge iff there is at least one k such that both T_i and T_j are contained in S_k.

Let A be some assignment and let n_{ij} be the number of tasks in T_j which are assigned to P_i in A:

$$n_{ij} = |A^{-1}(P_i) \cap T_j|, \quad 1 \leqslant i \leqslant m, \ 1 \leqslant j \leqslant h.$$

Let C_i be the total communication cost of A within the tasks in T_i and let C_{ij} be the total communication cost of A between the tasks in T_i and the tasks in T_j, $j \neq i$. We have clearly

$$Com(A) = c_0 \sum_{1 \leqslant i \leqslant h} C_i + c_0 \sum_{\{i,j\} \in E(G)} C_{ij}$$

with

$$C_i = \binom{n_i}{2} - \sum_{j=1}^{k} \binom{n_{ij}}{2},$$

(since the number of pairs of tasks in T_i is $\binom{n_i}{2}$ and the sum in the above counts the pairs which have no communication cost in A) and, for $\{i,j\} \in E(G)$,

$$C_{ij} = n_i n_j - \sum_{1 \leqslant k \leqslant m} n_{ki} n_{kj},$$

(since there are $n_i n_j$ distinct pairs picking a task in T_i and another in T_j and again the sum counts the pairs which have no communication cost).

Note that the communication cost of the assignment A depends only on the n_{ij}. Computing the minimum execution cost for given n_{ij} is again a Minimum Cost Flow Problem. Let us describe the corresponding network.

Network 3: Once again, there are task nodes and processor nodes. To each processor P_i, there correspond now h nodes p_{ij} each of which is linked to the task node θ_k by an arc with capacity 1 and cost e_{ki}. Moreover, p_{ij} is connected to the sink by an arc of capacity n_{ij}.

Again we will select the loads n_{ij} giving the overall minimum cost.

Let $n_j = |T_j| = \sum_{i=1}^{m} n_{ij}$. The number of choices for the n_{ij} for fixed j is the number of ordered partitions of n_j into h parts. It does not exceed $\binom{n_j-1}{h-1}$. Therefore, the time complexity is at most, up to a constant factor,

$$n^{5/2} \prod_{j=1}^{k} \binom{n_j - 1}{h-1} \leqslant n^{5/2}(n-1)^{(h-1)k} = \mathrm{O}(n^{h+3/2}) = \mathrm{O}(n^{2^k+1/2}).$$

4. The case where the communication graph is the union of several cliques and complete bipartite graphs

Assume now that the communication graph is the (edge) union of several cliques and complete bipartite graphs. This is the same as saying that the communication graph is defined by a partition of the set of tasks into a bounded number of subsets $\{T_1, T_2, \ldots, T_k\}$ where each T_i spans either a clique or the empty graph, and for each pair $i, j, i \neq j$, either none or all the edges between T_i and T_j are present. Let us denote by K the set of pairs $\{i,j\}$, $i \neq j$, such that the later case occurs and by L the set of indices i such that T_i is a clique. Let A be some assignment and let again n_{ij} be the number of tasks in T_j which are assigned to P_i in A. Now we have that

$$Com(A) = c_0 \sum_{i \in L} \left(\binom{n_i}{2} - \sum_{j=1}^{k} \binom{n_{ij}}{2} \right) + c_0 \sum_{\{j,l\} \in K} \left(n_j n_l - \sum_{j=1}^{k} n_{ij} n_{il} \right).$$

Thus, the communication cost of the assignment A depends again only on the n_{ij} and computing the minimum execution cost for given loads n_{ij} is again a minimum cost flow problem.

Acknowledgements

We thank an anonymous referee for suggestions which led to a substantial improvement of this paper. We also thank Claire Kenyon for helpful comments.

References

[1] A. Billionnet, M.C. Costa, A. Sutter, An efficient algorithm for the task allocation problem, J. ACM 39 (3) (1992) 502–518.
[2] S.H. Bokhari, A shortest tree algorithm for optimal assignments across space and time in a distributed processor system, IEEE Trans. Software Engrg. SE-7 (6) (1981).
[3] E. Dalhaus, D.S. Johnson, C.H. Papadimitriou, P. Seymour, M. Yannakakis, The complexity of multiway cut, in: Proceedings of the 24th ACM STOC, 1992, pp. 241–251.
[4] M. Grötschel, L. Lovász, A. Schrijver, Geometric Algorithms and Combinatorial Optimization, Springer, Berlin, 1988.
[5] M. Lamari, W. Fernandez de la Vega, The module allocation problem; an average case analysis, irregular 96, Santa Barbara, CA, Lecture Notes in Computer Science, Vol. 1117, Springer, Berlin, 1996, pp. 307–312.
[6] V.F. Magirou, An improved partial solution to the task assignment and multi-way cut problems, Oper. Res. Lett. 12 (1992) 3–10.
[7] V.F. Magirou, J.Z. Milis, An algorithm for the multiprocessor assignment problem, Oper. Res. Lett. 8 (1989) 351–356.
[8] V. Mary Lo, Heuristic algorithms for task assignment in distributed systems, IEEE Trans. Comput. TSC-37 (1988) 1384–1397.
[9] C.H. Papadimitriou, M. Yannakakis, Optimization, approximation and complexity classes, J. Comput. System Sci. 43 (1991) 425–440.
[10] H.S. Stone, Multiprocessor scheduling with the aid of network flow algorithms, IEEE Trans. Software Engrg. SE-3 (1) (1977) 85–93.